

# Task Interaction in an HTN Planner

Ilče Georgievski, Alexander Lazovik, and Marco Aiello

Distributed Systems Group  
Johann Bernoulli Institute for Mathematics and Computer Science  
University of Groningen  
{i.georgievski, a.lazovik, m.aiello}@rug.nl

**Abstract.** Hierarchical Task Network (HTN) planning uses task decomposition to plan for an executable sequence of actions as a solution to a problem. In order to reason effectively, an HTN planner needs expressive domain knowledge. For instance, a simplified HTN planning system such as JSHOP2 uses such expressivity and avoids some task interactions due to the increased complexity of the planning process. We address the possibility of simplifying the domain representation needed for an HTN planner to find good solutions, especially in real-world domains describing home and building automation environments. We extend the JSHOP2 planner to reason about task interaction that happens when task's effects are already achieved by other tasks. The planner then prunes some of the redundant searches that can occur due to the planning process's interleaving nature. We evaluate the original and our improved planner on two benchmark domains. We show that our planner behaves better by using simplified domain knowledge and outperforms JSHOP2 in a number of relevant cases.

**Keywords:** HTN planning, task interaction, phantomization, domain knowledge

## 1 Introduction

Hierarchical Task Network (HTN) planning is a well-known and widely used artificial intelligence planning technique. HTN planners are provided with a set of goal tasks that have to be repeatedly decomposed until primitive tasks are reached. Such task decomposition is based on the heuristics or subplans contained in the domain knowledge. Well written heuristics or methods can significantly reduce search space and help planner to find an efficiently executable plan. The main advantage of using HTN technique is the way of writing the heuristics which can be seen as recipes that fit well with human being thinking. Using heuristics during the planning process means that there should be different interactions between tasks.

One type of interaction describes how task's effects are already achieved by other task(s) at some place in the task network. Most of the HTN planners solve this task interaction by using the notion of a *phantom task* [10]. Among these planners, a very popular implementation is SHOP2, a sound and complete

planner [7]. The SHOP2 planner can reason if certain effect has been achieved only if the domain knowledge contains such a phantom task (or several phantom tasks). However, this type of reasoning depends on the domain’s writer ability and experiences to identify and encode such task interaction and not on the planner’s reasoning capabilities. Planners as SHOP2 avoid some task interactions because they can increase the complexity of the planning process.

In general, writing knowledge for controlled/synthetic domains is easier than for real-world ones. Even more, it is cumbersome to identify interactions between tasks in real-world settings. As an illustration, we introduce GreenerBuildings<sup>1</sup> project. GreenerBuildings’s objective is to develop generic principles and an energy-aware framework that utilizes human activity and context recognition techniques to adapt buildings for energy saving. To date, buildings involve many manual tasks, as switching lights and appliances to seasonal changes in the heating systems. Nevertheless, substantial energy savings in buildings can be achieved by globally switching and regulating installations and appliances to actual needs. Hence, the approach introduces concepts for self-powered sensing, processing and actuation in large distributed networks that dynamically minimize energy consumption. Such distributed networks are planned to have approximately 1000 devices and, thus, a very large number of operators (if we assume that each device represents appropriate operator). Encoding such a complex domain is very tedious, especially when optimal solutions are essential as the goal of the GreenerBuildings is to be energy aware. Operators can be associated with costs that reflect the amount of energy their execution requires. Considering this information, the planner has to choose a decomposition that corresponds to the minimum energy consumption. Even more, this decomposition should be optimal i.e. it should not contain redundant operators. Hence, writing phantom tasks is not the best solution. Taking task interactions into account, such a redundancy can be avoided in simple and elegant way.

In this paper, we extend the best-known JSHOP2<sup>2</sup> to reason explicitly over the above-mentioned task interactions. We transfer some of the domain expressivity into the planning process itself with the goal of keeping the domain representation as simple and compact as possible. This is especially useful when the domain writer is not familiar with some specific-purpose representation, e.g., a phantom task. We introduce a task-to-task matching, which finds a task that is accomplished and reasons that current task’s effects are achieved by already accomplished task and that the current task can be ignored, if and only if the effects are still holding. By adding the ability to identify and solve such task interaction, the planner can also control the search space by avoiding some re-

---

<sup>1</sup> GreenerBuildings is an Information and Communication Technologies project funded under the European Seventh Framework Programme on Engineering of Networked Monitoring and Control Systems and Wireless Sensor Networks and Cooperating Objects. More information can be found on <http://greenerbuildings.eu>

<sup>2</sup> A Java implementation of SHOP2. JSHOP2 is available as open source on <http://sourceforge.net/projects/shop/files/JSHOP2/>

dundant paths. Moreover, we show that the planner performs very good in cases when JSHOP does not.

The rest of the paper is organized as follows. Section 2 defines the basic HTN terms and the pantomization process. Section 3 introduces the algorithm in details and an example of applying it. Next, Section 4 shows the implementation and the evaluation on two benchmark domains. Section 5 discusses the addressed issue and reviews related work. Finally, we finish with concluding remarks in Section 6.

## 2 Background

### 2.1 HTN Planning

Intuitively, an HTN planning technique can be viewed as an extension of the classical planning approach. The objective of an HTN planner is not to achieve a set of goals but instead to perform a set of (goal) tasks. An HTN domain contains, besides the set of operators, a set of high-level descriptions called methods. Each method can be decomposed into a task network or a set of tasks with ordering constraints between them. Each task should satisfy certain conditions. An HTN planning problem includes initial task network that is decomposed into a sequence of operators i.e. a plan.

*Example 1.* Consider the following example of high-level descriptions. We have two HTN methods, one for adapting the office for living atmosphere (*adjust-office*) and another method for adjusting the work desk (*adjust-desk*). Task *adjust-office* can be decomposed into task network of three subtasks *set-AC*, *turn-on-light*, *turn-on-music*. Task *adjust-desk* contains decomposition of two subtasks, namely operators *turn-on-light* and *start-computer*.

Next, we provide formalism for an HTN planning that follows the one of Ghallab *et al.* [4].

A *task* is an expression formed by a *task symbol* and a set of terms that can be constants or variables. If the task symbol is an operator symbol, then the task is *primitive*; otherwise, the task is *nonprimitive*. In our example, *set-AC*, *turn-on-light*, *turn-on-music*, and *start-computer* are primitive tasks, while *adjust-office* and *adjust-desk* are nonprimitive tasks.

**Definition 1.** A *task network* is a pair  $w = (U, C)$  where  $U$  is a set of task nodes and  $C$  is a set of constraints. Each task node  $u \in U$  contains a task  $t_u$ . The task network is primitive if all of the tasks are primitive; otherwise,  $w$  is nonprimitive.

Each constraint in  $C$  specifies a condition that must be satisfied by every plan that is a solution to the planning problem. Examples of constraints are *precedence constraint*, *before-constraint*, *after-constraint* and *between-constraint*.

Considering the above example, we could have a task network where  $U = \{u_1, u_2, u_3\}$ ,  $u_1 = \text{set-AC}$ ,  $u_2 = \text{turn-on-light}$ , and  $u_3 = \text{turn-on-music}$ , and  $C$

contains a precedence constraints, for example, that  $u_1$  must occur before  $u_2$  and  $u_2$  must occur before  $u_3$  and a before-constraint such that the air conditioning system is serviceable before  $u_1$ .

**Definition 2.** An *HTN method*  $m$  is a 4-tuple  $(name(m), task(m), subtasks(m), constraints(m))$  respectively referring to the method's name, a nonprimitive task, and the method's task network containing subtasks and constraints.

The descriptive name for the nonprimitive task *adjust-office* could be *adjust1* with subtasks and constraints described above.

A method instance  $m$  is applicable in a state  $s$  if its preconditions are satisfied in the  $s$ . A method instance  $m$  is relevant to task  $t$  if there is a substitution  $\sigma$  such that  $\sigma(t) = task(m)$ .

**Definition 3.** An *HTN planning domain* is a pair  $D = (O, M)$ , where  $O$  is a set of operators and  $M$  is a set of methods.

Taking into account our example, we could define a domain with five operators and two methods.

An operator  $o$  is an action described by a 3-tuple  $(name(o), preconditions(o), effects(o))$  referring to operator's name, preconditions and effects, respectively.

**Definition 4.** An *HTN planning problem* is a 4-tuple  $P = (s_0, w, O, M)$ , where  $s_0$  is the initial state,  $w$  is the initial task network and pair  $(O, M)$  is the planning domain.

Finally, a plan  $\pi = \langle a_1, a_2, \dots, a_n \rangle$  is a solution for planning problem  $P$  if there is a sequence of task decompositions that can be applied to  $w$  to produce a primitive task network  $w'$  such that  $\pi$  is a solution for  $w'$  (considering that  $w$  is a nonprimitive).

For example, the plan  $\pi = \langle set-AC, turn-on-light, start-computer, turn-on-music \rangle$  could be a solution for the problem of achieving both tasks, *adjust-office* and *adjust-desk*.

## 2.2 Phantomization

A good and optimal plan should not contain redundant primitive tasks. There are different aspects of how can unnecessary plan steps be reduced. One way to accomplish reducing is during the planning process considering certain domain descriptions.

Tate in [10] has introduced the term *phantom task* as a way of treating the task as already achieved at some point in the network by other task(s). The phantom task can be accomplished by **doing nothing**, if this task is placed in the network at a point where its effect is still holding. This type of task reduction is known as *phantomization*. Young *et al.* [12] have stated that the idea of phantomization is key to the appropriate performance of the planners that perform task decomposition. The advantage of using phantomization is the planner's ability to reason which tasks are unnecessary, and, therefore, to produce more efficient plans. The weak point is its identification and encoding into the domain representation.

### 3 Task Interaction

We address the possibility of diminishing the tedious writing of effective domain knowledge by introducing an enhanced reasoning over one type of task interaction and demonstrating it by extending currently the most popular simplified HTN planner JSHOP2. The reasoning is performed by checking whether the current task's effects are already achieved by other same named task, and are still holding at the current state. In the case where these effects are still holding, the planner reasons that this task is redundant, avoids applying it and continues with the planning process. By enabling this task interaction, the planner also has to control the search space as this interaction can happen in different levels of task interleaving which can lead to redundant searches or plans (if such exist).

In this way, the planner is enhanced with the ability to find a plan even when the domain writer does not provide highly efficient (alternative) methods. The remainder of the section describes the planning algorithm, and a simple problem example.

#### 3.1 Algorithm

Our algorithms outlined in Alg. 1 and Alg. 2 are high-level descriptions incorporated in the existing JSHOP2 planning and interleaving algorithms. Algorithm 1 takes as input a planning problem  $(s_0, w_0, D)$ , as defined in the previous section, and an agenda  $A$ . Algorithm 2 takes as input a task list  $w_i$ , and an agenda  $A$ .

---

**Algorithm 1** task interaction

---

```
1: Call interleave to choose an appropriate task  $t \in w_i$ 
2: if  $t$  is a primitive task then
3:   if  $t$  is applicable in the current state  $s_i$  then
4:     add the  $t$ 's effects to the  $A$ 
5:   end if
6: else if  $t$  is a nonprimitive task then
7:   if same named method has been previously reduced then
8:     get it's subtasks and call this algorithm recursively
9:   end if
10:  if  $t$  is reducible in the current state  $s_i$  then
11:    add  $t$  to the reduced methods list
12:  end if
13: end if
```

---

Let us examine the algorithms in detail. Algorithm 1 starts with the interleaving step which calls helper algorithm Alg. 2, as noted in line 1 (Alg. 2 shows only incorporated steps in the existing interleaving mechanism of JSHOP2). Algorithm 2 prunes each primitive task that is already applied (i.e. a step in the potential plan) and its effects are elements of  $A$ , as it is shown in lines 1-5.

Agenda  $A$  contains all facts that are holding up to the  $i$ -th state. When these conditions are fulfilled we say that task  $t$  is matchable. Formal definitions follow.

**Definition 1.** Let  $s_i$  be the current state. Agenda  $A$  consists of a set of logical atoms which values are accurate in the  $s_i$  state.

**Definition 2.** Let  $t$  be the current primitive task,  $t_a$  already applied primitive task,  $s_i$  the current state, and  $A$  is the agenda. Task  $t$  is matchable with  $t_a$  if and only if  $t$  and  $t_a$  denote same operator instance, and  $t_a$ 's effects are elements of  $A$  in the state  $s_i$ .

---

**Algorithm 2** *interleave*

---

```

1: if  $t \in w_i$  is a primitive task then
2:   if same named operator is already applied and  $t$ 's effects are in  $A$  then
3:      $t$  is matchable and do not interleave it
4:   end if
5: end if

```

---

Planning continues depending on whether the chosen task is primitive or nonprimitive. If the chosen task  $t$  corresponds to a primitive task, which is applicable in the state  $s_i$ , its effects are added to the list of logical atoms  $A$ , as noted in lines 2-5. When the chosen task  $t$  is a nonprimitive task (see line 6), JSHOP2 skips the method's branches for which bindings do not exist. However, when we consider task interaction we should check also some of these branches, in particular, those ones that have been already successfully instantiated (i.e. their primitive subtasks are part of the potential plan). Therefore, in lines 7-9 we add the logic to the algorithm that reduces this type of branch and call the algorithm recursively with branch's subtasks as task list  $w_{i+1}$ , which matchability can be further checked. Formal definition follows.

**Definition 3.** Let  $t$  be the current nonprimitive task,  $t_j$  is the  $j$ -th branch of  $t$ , and  $s_i$  is the current state. Task  $t$  is reducible into branch  $t_j$  if and only if  $t_j$  is instantiated in state  $s_k$ , where  $k < i$ .

JSHOP2 is a partial-ordered planner, which produces all possible combinations of tasks' sequences. Previously identified task interaction enables additional interleaving steps between tasks. Indeed, when the interaction has already happened many of the interleaving steps are not necessary as they produce redundant searching. Hence, we have added a control ability to the algorithm that prunes these steps. For instance, if the planner finds a plan at some point after successful task interaction and backtracks to try other combinations, without controlling the search it will find a number of plans which are equal as the first found one.

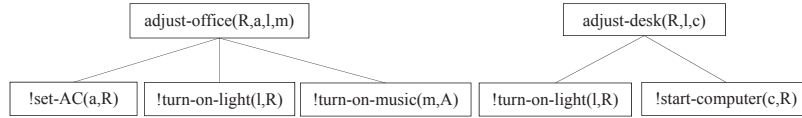
The algorithm continues searching for other possible branches of  $t$  for which exist appropriate binding in the  $i$ -th state. Thus, when certain method's branch

is reducible in the current state  $s_i$ , method's branch is added to the list of reduced methods (see line 10).

Some of the Alg. 1's steps are excluded from the sketch. They are the same as the those in JSHOP2.

### 3.2 An Example

Consider the example from Sec. 2. Say that we want to perform both tasks i.e. two goal tasks: preparing the work desk in office  $R$  and adjusting office  $R$  after some time being empty. More specifically, the task of adjusting the desk can be decomposed into two subtasks of turning on the light  $l$  and starting the computer  $c$ . Task of adjusting the office can be decomposed into three subtasks of setting the air conditioning system  $a$ , turning on the light  $l$  and switching on the music system  $m$  (see Fig. 1). One of the most effective solutions is when the air conditions are comfortable, the light is turned on, the music is playing and the computer is started and ready for work.



**Fig. 1.** Example of two tasks decompositions

Listing 1.1 outlines the above methods described with the JSHOP2 notation. Methods `adjust-desk` and `adjust-office` do not contain any preconditions to keep representation as simple as possible. Methods' descriptions are the same as in the graphical design except that they are generalized for any terms by using variables.

```

(:method (adjust-office ?r ?a ?l ?m)
  ()
  ((!set-AC ?a ?r) (!turn-on-light ?l ?r) (!turn-on-music ?m ?r)
  )
)
(:method (adjust-desk ?r ?l ?c)
  ()
  ((!turn-on-light ?l ?r) (!start-computer ?c ?r))
)

```

**Listing 1.1.** Simple Methods' Descriptions

We can now examine the situation when the algorithm is on the right way of finding a good solution. Let us assume straightforward applying of the first two operators `(!set-AC a R)` and `(!turn-on-light l R)` from method `(adjust-office R a l m)`. We should note that their effects are added to

the agenda at the applying point. Process continues by interleaving the method (`adjust-desk R l c`) and reducing it to its task network. Method's first sub-task is (`!turn-on-light l R`) which we assumed that is already a part of the potential plan. Thus, the algorithm reasons that this task is already achieved and that its effect (the light  $l$  is on) is still holding. Therefore, the algorithm is allowed to prune the task from interleaving and continues by processing the rest of available tasks. In few steps it finds the correct sequence of operators, i.e. the plan.

In contrast to our solution, JSHOP2 will not find a plan by having in mind the above domain description. In order to be able to find a solution, we have to improve the domain with more effective descriptions. In List.1.2 we enclose enhanced descriptions. As we can see, we have included additional method `light-helper` which has a decomposition representing a phantomization process of doing nothing when the light is already turned on. Comparing List.1.1 and List.1.2 is obvious that we make more simple and compact domain representation.

```
(:method (adjust-office ?r ?a ?l ?m)
  ()
  ((!set-AC ?a ?r)(light-helper ?l ?r)(!turn-on-music ?m ?r)
  )
)
(:method (adjust-desk ?r ?l ?c)
  ()
  ((light-helper ?l ?r) (!start-computer ?c ?r))
)
(:method (light-helper ?l ?r)
  (not (on ?l ?r) )
  (!turn-on-light ?l ?r)

  (on ?l ?r)
  ()
)
)
```

**Listing 1.2.** Enhanced Methods' Descriptions

## 4 Implementation and Evaluation

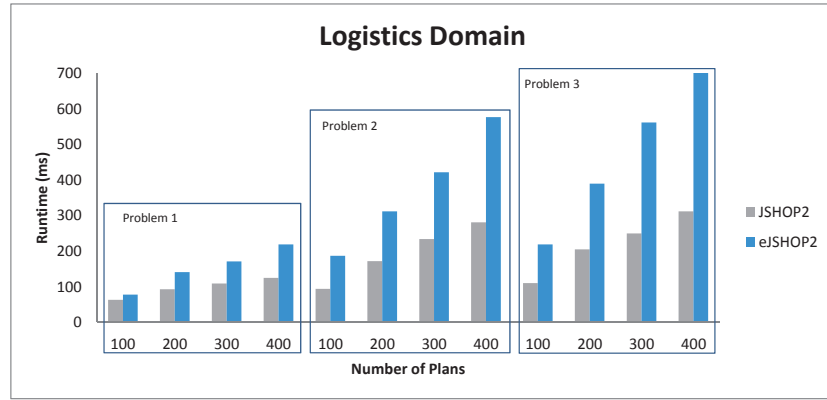
As mentioned above, we used SHOP2 Java implementation. We have evaluated our implementation on two benchmark planning domains, namely the Logistic domain which is available with the JSHOP2 source code and Dock-Worker Robot [4] which we adapted to fit with the JSHOP2 notation.

We have three objectives by performing evaluation: to show that our enhanced eJSHOP2 planner is able to find solutions with simplified domain description, to show that its performance is nevertheless reasonable compared



to JSHOP2 planner despite added complexity to the planning process, and to present that our planner can perform better in some cases.

Figure 2 shows the results by testing the planners in the Logistics domain. We have created three problems with gradual complexity. Problem 1 includes two locations and two packages that have to be transferred from one location to the other. Problem 2 examines the situation with three locations and four packages, and Problem 3 tests 4 locations with six packages to transfer. For each problem the planners have tried to find 100, 200, 300 and 400 plans. As we can see, in every case JSHOP2 has better performance. The reason for such a behavior is due to the reasoning in the preconditions and not having additional search spaces to look for a plan, while in our implementation planning includes more chances for additional search space and backtracking along with it.



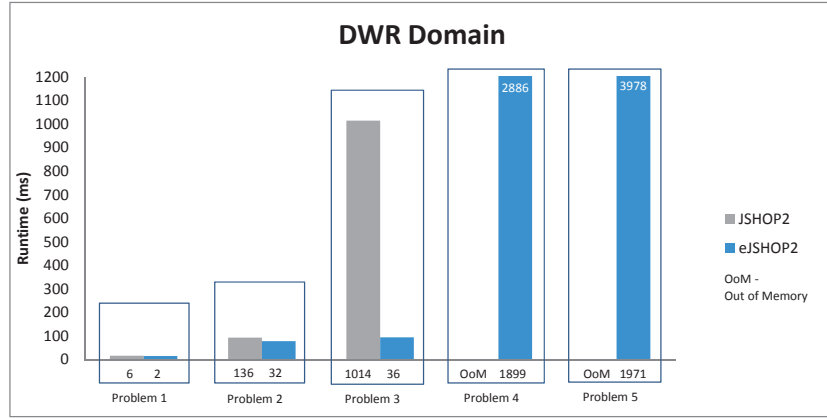
**Fig. 2.** Runtime Measurement on the Logistics Domain

Figure 3 shows a little different results. Planners have been tested on the Dock-Worker Robots domain. There are five different problems on which planners have tried to find all possible solutions. JSHOP2 planner has found more plans due to its inability to prune the search space that promises a number of redundant plans as a result of the combinatorial nature of the interleaving process. In a slightly more complex problems it easily runs out of memory. On the contrary, our eJSHOP2 planner performs faster as a result of the possibility to prune some of the ways that can lead to redundant searches.

All experiments were performed on a Windows-based machine with 3 GB of memory and a 2.00 GHz Intel Core 2 Duo processor.

## 5 Discussion and Related Work

The main advantage of HTN planning is its ability to deal with very large and complex problem domains. However, HTN requires experienced domain writers



**Fig. 3.** Runtime Measurement on the Dock-Worker Robots Domain

to provide the planner with the descriptions that it needs to plan. Especially, this is a case with HTN planners that know everything about the state of the world at each planning step. JSHOP2 produces a plan based on such a principle and executes the plan latter. Worthily mentioning is that this kind of planners guarantee that a plan will be found if all the necessary requirements are provided. Indeed, we have to write powerful methods and to describe all possible situations that might happen in the specific domain. Therefore, we have to assume that we exactly know how these descriptions will affect on the planning process and the world state. However, we question if there is a possibility to transfer some of the expressivity of the domain representation to the planning process itself.

We have shown that in certain cases the domain representation can be simplified. Although this contribution has slowed down the performance of the planner, there are cases when this implementation can be useful. Planners as JSHOP2 avoid task interactions since they can expand the search space and increase the complexity of the planning process.

SHOP2 is sound and complete planner [8] or it can produce a correct plan if all necessary requirements are fulfilled i.e. a set of methods and operators is able to generate a solution for a problem. Understanding the completeness and soundness of our extension is straightforward as by simplifying the domain representation we enhanced the algorithm to be able to deal with such simplification.

Usually planners' performances are evaluated on standard benchmark domains. However, there is no simple and absolute way to judge the efficiency of a planning system. One should understand the computational complexity of planning in a particular domain in order to be able to asses the efficiency of the planner. If no planning system performs well in a given domain, does it mean that all planners are bad, or is it domain naturally hard? Such questions and answers are addressed in [5].

The idea of reducing redundant tasks is not new. Past 30 years a lot of research has been done in finding a way of reducing unnecessary plan steps.

One way to accomplish this idea is during the planning process. Several task decomposition planners support this way of reduction [9] [10] [11]. All planners use the domain expressivity to reason about the redundancy and, to the best of our knowledge, no one has incorporated that ability to the planner itself.

As mentioned, Tate [10] has introduced the phantom task that can be accomplished by doing nothing, if this task is placed in the network at a point where its effect is still holding. The phantomization process is used in the framework for plan modification and reuse [6]. It is stated that when the task  $t$  is of the form  $achieve(C)$ , and  $C$  can be achieved directly by using the effects of some other task  $t_c \in T$ , where  $T$  is a set of tasks, then  $t$  becomes a phantom task and its reduction becomes  $\langle \{phantom(C)\}, \emptyset, \emptyset \rangle$ . All these task reduction schemas are given to the planner *a priori* as part of the domain specification.

In addition, [1] has proposed an idea to merge tasks with similar actions. To merge two tasks means that their actions must “match” in the sense that they differ only in the slots where one or both have anonymous constants. This idea has no formalism nor practical implementation, thus, it is impossible to compare.

Foulser *et al.* [3] have proposed a formalism and both optimal and efficient heuristics algorithms for finding minimum-cost merged plans. In their formal theory a set  $\Sigma$  of operators is mergeable with a (merged) operator  $\mu$  if and only if  $\mu$  can achieve all the useful effects of the operators in  $\Sigma$ , and  $\mu$ 's preconditions are subsumed in the preconditions of  $\Sigma$ , and the cost of  $\mu$  is less than the cost of  $\Sigma$ . Although our idea is similar to theirs, we use less constrained task to task interaction in an HTN planning. Hence, we consider two types of tasks, methods and operators.

In the multiagent environment, Cox and Durfee [2] have described an algorithm that uses a merging approach to help agents remove redundant plan steps while at the same time can preserve their autonomy. Basically, their approach remove redundant steps from plans instead of not adding them at all.

## 6 Concluding Remarks

We presented an algorithm with the ability to reason about one type of task interaction. Most of existing systems use domain's methods to deal with this kind of interaction. We included the phantomization process as part of the planning process itself.

We have extended JSHOP2 planner to be able to identify and process already existing effects. Thus, the planner is capable of pruning some redundant searches. This was not case with previous systems as this cannot be included as ability into the domain heuristics.

In future work, there is possibility to extend this task-to-task matching into a task-to-tasks matching. In this case, it is possible to optimize some plans even more by reducing the number of steps necessary to have a solution to the planning problem. We believe that there is an opportunity to include more task interactions into the planning process to the planners as JSHOP2.

**Acknowledgment.** We thank our colleague Eirini Kaldeli for the helpful discussions. This work is supported by the GreenerBuildings Project funded under the European Seventh Framework Programme (FP7) with EC contract number INFSO-ICT-258888.

## References

1. Charniak, E., McDermott, D.: Introduction to artificial intelligence. Addison-Wesley series in computer science, Addison-Wesley (1986)
2. Cox, J.S., Durfee, E.H.: Discovering and exploiting synergy between hierarchical planning agents. In: AAMAS. pp. 281–288 (2003)
3. Foulser, D.E., Li, M., Yang, Q.: Theory and algorithms for plan merging. *Artif. Intell.* 57(2-3), 143–181 (1992)
4. Ghallab, M., Nau, D.S., Traverso, P.: Automated planning - theory and practice. Elsevier (2004)
5. Helmert, M.: Understanding Planning Tasks: Domain Complexity and Heuristic Decomposition, Lecture Notes in Computer Science, vol. 4929. Springer (2008)
6. Kambhampati, S., Hendler, J.A.: A validation-structure-based theory of plan modification and reuse. *Artif. Intell.* 55(2), 193–258 (1992)
7. Nau, D.S., Au, T.C., Ilghami, O., Kuter, U., Murdock, J.W., Wu, D., Yaman, F.: Shop2: An htn planning system. *J. Artif. Intell. Res. (JAIR)* 20, 379–404 (2003)
8. Nau, D.S., Muñoz-Avila, H., Cao, Y., Lotem, A., Mitchell, S.: Total-order planning with partially ordered subtasks. In: IJCAI. pp. 425–430 (2001)
9. Sacerdoti, E.D.: A structure for plans and behavior. Ph.D. thesis, Stanford, CA, USA (1975), aAI7605794
10. Tate, A.: Generating project networks. In: IJCAI. pp. 888–893 (1977)
11. Wilkins, D.E.: Practical planning: extending the classical AI planning paradigm. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1988)
12. Young, R.M., Pollack, M.E., Moore, J.D.: Decomposition and causality in partial-order planning. In: AIPS. pp. 188–194 (1994)